LeetCode #260 Single Number III (查找数组中2个不成对的数)

Posted on 2015/08/19 by NeoShell —

题目：

Given an array of numbers `nums`*, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once.*

For example:

Given `nums = [1, 2, 1, 3, 2, 5]`*, return* `[3, 5]`*.*

Note:

1. *The order of the result is not important. So in the above example,* `[5, 3]` *is also correct.*
2. *Your algorithm should run in linear runtime complexity. Could you implement it using only constant space complexity?*

分析：

数组中除了2个数以外，其他所有的数都是成对的。要求把这2个数找出来。

本题也用到了异或运算的性质：**A^A = 0, A^B^A = B**

首先遍历数组，将数组中所有的数进行异或之后，得到值就等于不成对的两个数的异或值。例如，对于[1,2,1,3,2,5]，异或所有的数得到6，这也正是其中不成对的两个数3和5的异或结果。

得到这两个数的异或结果有什么用的？怎样把他们分开？

我们知道，既然两个数不成对，那么他们之间至少存在一个二进制位是不同的。我们可以通过这个不同的位来区分这两个数。对于异或运算的结果，1表示原来的两个数在这一位上不同。现在找到异或结果的任何一个置为1的位即可。还是看刚才的例子，3和5的异或运算如下：

```
      3 (0011)
xor   5 (0101)
=     6 (0110)
```

现在我们想知道异或运算结果的某个为1的位。最简单暴力的方法就是遍历每一位，看是不是1。但是，我们有一个更tricky的方法：**一个数和它的相反数做与运算，得到的结果就是该数的最低的为1的位。**还是用刚才的例子(为了简洁，我们假设是4位二进制补码)：

```
     +6 (0110)
and  -6 (1010)
=       (0010)
```

现在我们就知道了，3和5在倒数第二位上有所不同。**再次遍历数组，只是这次我们根据倒数第二位上是0还是1，把数组中的数分成两组。这样可以保证3和5一定不在同一组。然后再对2组分别进行异或运算，即可得到最后的结果。**

代码：

Java

Java

```java
public class Solution {
    public int[] singleNumber(int[] nums) {
        int[] res = new int[2];
        int temp = 0;
        for(int n : nums) temp ^= n;
        temp &= -temp; // Get the last set bit
        for(int n : nums){
            if((n & temp) == 0) res[0] ^= n;
            else res[1] ^= n;
        }
        return res;
    }
}
```

相关文章：

《LeetCode #136 Single Number》

《LeetCode #137 Single Number II》

参考文章：

https://leetcode.com/discuss/52351/accepted-java-space-easy-solution-with-detail-explanations